(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 813 132 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
17.12.1997 Bulletin 1997/51

(51) Int. Cl.[6]: **G06F 1/00**, G06F 9/46, H04L 29/06

(21) Application number: **97303443.2**

(22) Date of filing: **20.05.1997**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **11.06.1996 US 661517**

(71) Applicant:
International Business Machines Corporation
Armonk, N.Y. 10504 (US)

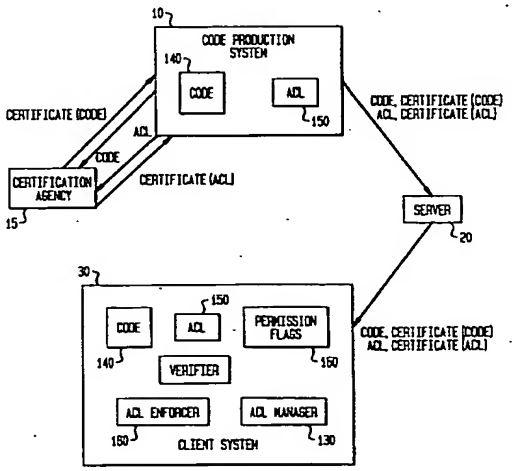
(72) Inventors:
- **Dan, Asit**
  West Harrison, New York 10604 (US)
- **Ramaswami, Rajiv**
  Ossining, New York 10562 (US)
- **Sitaram, Dinkar**
  Yorktown Heights, New York 10598 (US)

(74) Representative: **Waldner, Philip**
IBM United Kingdom Limited,
Intellectual Property Department,
Hursley Park
Winchester, Hampshire SO21 2JN (GB)

(54) Support for trusted software distribution

(57) A form of authentication is provided wherein a trusted third party signs a certificate to identify the author of a program and to secure its integrity. The program code is encapsulated or otherwise associated with the certificate and an access control list (ACL). The access control list describes the permissions and resources required by the code. An enforcement mechanism which allocates system permissions and resources in accordance with the ACL. In a preferred embodiment, a code production system communicates with a certification agency, which is a trusted third party. The certification agency issues a certificate for the code and a certificate for the access list of that code. Once the certificate is issued it is not possible for any party to modify the code or access list without invalidating the certificate. The code and its ACL, along with their certificates are stored on a server. A client downloading the code or access list can verify the integrity of the code/access list and the system can enforce the access list such that the permissions and resources are not exceeded.

FIG. 1



EP 0 813 132 A2

Description

The present invention relates generally to delivery of software through distribution systems such as networks.

Software distribution is a major industry currently. Software is today distributed over diskettes, CDROMs and increasingly, over networks. In addition to simply downloading code from remote sites over the Internet, today clients can download and run applets from servers, a paradigm proposed in the Java programming language or in other languages such as Telescript.

A significant concern with code obtained from elsewhere is that of security. For example, in the Unix operating system, the code will run in the client's shell with all the client's privileges, including access to all his files, as well as possibly send mail, attempt illegal break ins etc. Java attempts to solve this problem for applets by running applets in a very restrictive environment. As a result, the usefulness and functionality of Java applets is limited. Java applications on the other hand rely on the security provided by the operating system and do not come with any degree with authentication. They thus pose the same security problems as any other code.

A form of authentication is proposed in "Trusted Distribution of Software Over the Internet", A.D. Rubin, Internet Society Symposium on Network and Distributed Security, 1995. Here a trusted third party signs a certificate to identify the author of a program and to secure its integrity. While this allows a client to verify the authenticity of the code, it does not specify a flexible set of permissions associated with the code nor does it provide an automated method for the client to enforce these permissions.

In a preferred embodiment, a code production system communicates with a certification agency, which is a trusted third party. The certification agency issues a cerificate for the code and a certificate for the access list of that code. Once the certificate is issued it is not possible for any party to modify the code or access list without invalidating the certificate. The code and its ACL, along with their certificates are stored on a server. A client downloading the code or access list can verify the integrity of the code/access list and the system can enforce the access list such that the permissions and resources are not exceeded.

According to one aspect of the present invention there is provided a method for distributing program code comprising the steps of providing to a recipient system a trusted third party certification, the trusted third party certification including a computer readable description of resources and permissions required for verified non-harmful operation of the code.

According to a second aspect of the present invention there is provided a method for distributing program code comprising the steps of:

providing a recipient system with an encrypted trusted third party certification of the program code, the trusted third party certification being encapsulated with the program code and including a computer readable description of resources and permissions required for verified non-harmful operation of the code;
reading the certification by the recipient system;
determining the integrity of the certification by the recipient system; and
only after the integrity has been verified,
allocating resources and permissions of the recipient system in accordance with user selected options so as not to exceed the permissions specified in the certification; and
executing the program code in accordance with the allocating.

wherein the descriptions of resources required includes data describing both a quantity of each resource to be used by the code and a maximum rate of consumption of each resource by the code and wherein the descriptions of permissions required includes data describing specific facilities of the recipient system to be accessed by the code.

Figure 1 shows a block-diagram view of a code delivery and certification system according the preferred embodiment of the invention;
Figure 2 shows how the different parts of the client system operate together to perform the functions described in the embodiment;
Figure 3 shows the access control list (ACL);
Figure 4 shows the data structures used by the ACL enforcer;
Figure 5 shows how logical resource permissions are enforced by the ACL enforcer;
Figure 6 shows how physical resource limits are enforced by the ACL enforcer;
Figure 7 shows the pseudocode for the integrity verification, execution and enforcement initialization operations of the client system;
Figure 8 shows the pseudocode for the ACL manager; and
Figure 9 shows the pseudocode for the ACL enforcer.

Figure 1 shows a block-diagram view of a code delivery and certification system according to a preferred embodi-

ment of the invention. It includes of one or more code production systems (CPS) 10, a certification agency (CA) 15, one or more servers 20 and one or more client systems 30. The client systems can be coupled to the servers by way of a conventional Wide Area Network (WAN) or by way of a Local Area Network (LAN). The CPS has a piece of program code (code) 140 and an access list (ACL) 150 for that code that it wishes to get certified by the certification agency. The CA provides a public key K that is widely available and known to the client system. In addition the CA has a private key P that only it knows. To provide a certificate for the code the CA creates a certificate containing the code name and the cryptographic hash of the code and signs it using his private key. The certificate cannot now be changed without invalidating the CA's signature. Similarly the CA creates and signs another certificate for the ACL associated with that code (if desired there could be a single certificate for both the code and its access list). In the embodiment of Figure 1, the client systems receive the certificate, the ACL and the code by way of the WAN or LAN. It should be understood, however, that the client systems can also or alternatively receive certified code by way of a removable storage media read by local importation device such as a floppy or optical disk drive.

The client system shown in Figure 2 includes a verifier 110, ACL manager 130, executor 170, ACL enforcer 180 and a client interface 190. The client system also includes a conventional CPU and associated control logic 192, a communication controller/network interface 194, a CD-ROM and/or floppy disk subsystem 196 and other resources such as network access modules, a display subsystem and various special purpose adapters. Those of skill in the art will recognize that the client system includes a number of additional, conventional components which will not be described in detail here.

The ACL manager 130 and the ACL enforcer 180 are preferably embodied as program code, the operation of which is described in more detail below. The executer is a conventional part of the client's operating system (not shown) which is used to execute the imported code on the client system. The client interface 190 can be embodied as a front of screen graphical user interface which provides the client communication with the ACL manager (e.g. it enables the client to tell the ACL manager 130 to allow or disallow or control program access to specified resource). The verifier 110 is preferably embodied as program code including a decryption module which is used to verify the authenticity of the imported code, including the access control list. The verifier also includes a hashing module which checks the integrity of the imported code and ACL.

Upon downloading the code/ACL and its certificate (100), the verifier (110) first checks to see if the CA's signature on the certificate is valid (using CA's known public key). The verifier then computes the cryptographic hash of the code/ACL and verifies that it matches the value in the certificate. If the signature is not valid or the hash does not match, the code and ACL are rejected (120). If the verification is ok, the ACL manager (130) is invoked. The ACL manager displays the ACL (described below) to the client via the client interface (190) and ascertains whether the client wishes to allow or disallow the individual items in the ACL. The ACL manager stores the code (140) as directed by the client via the client interface and stores the ACL (150) together with permission flags (160).

The resources that can have their access controlled by the ACL enforcer include both logical resources such as file-systems, specific files and system calls, as well as physical resources including such things as the disk space, disk access, main memory allocation and access to various system controllers and adapters. For access to logical resources (client system facilities), the permission flags 160 indicate whether individual items are allowed or not; for access to physical resources, the permission flags can be used to indicate the maximum allowable quantity or rate of consumption.

Figure 2 also shows how the different parts of the client system operate together. In a multi-user client system, each user may have their own set of permission flags. The ACL may also contain environment variables; changing the environment variables during execution allows individual users to customize the access privileges. The ACL and permission flags are stored in a secure area; reading or updating this area is a privilege enforced by the ACL enforcer (180).

The ACL manager may be invoked at any time by the client to display or change the permissions of an ACL. The code is run by the executor (170). Before allowing access to any resource, the executor invokes the ACL enforcer for checking the validity of the access. This is achieved by the executor 170 inserting traps in the code to a verification routine that invokes the ACL enforcer. The operation of the ACL enforcer is described in more detail later.

The system enables the code and its ACL to be downloaded separately or together as needed. For instance, the CPS may want to provide the ACL free of charge to all clients but charge for the actual code.

Figure 3 shows the access control list. The ACL consists of two parts: the Physical Resources Table (PRT) 200 containing the physical resources required by the code and the Logical Resources Table (LRT) 250 containing the permissions and logical resources required by the code.

The PRT 250 contains a row for each resource containing the physical resource name (PRN) 205, the resource attribute 210, the maximum consumption rate 215 and the maximum amount 220. The resource attribute 210 is used when a physical device or resource disk has multiple attributes, such as space and number of I/Os for storage devices. The maximum consumption rate 215 and the maximum amount 220 are the maximum allowable consumption rate and maximum allowable consumption of the resource and attribute respectively.

The LRT 250 contains a row for each call to an external routine required by the code (referred to as a logical resource). Each row contains the logical resource name (LRN) 255, and a parameter list 260 that points to a list of

parameter entries. Each parameter entry 265 specifies a set of valid parameter ranges; i.e. a set of values for each parameter that are valid in combination, together with a field nextPE 280 that points to the next parameter entry. The parameter range for each parameter contains two fields - the parameter type 270 and the parameter value 275 that specifies the valid range for that parameter. For string parameters, the parameter type 270 is STR and the parameter value 275 is a list of regular expression that specify valid forms for the string. For integer parameters, the parameter type 270 is INT and the parm. value 275 is a list of integer ranges. The parm. value 275 may include the names of environment variables, in which case the environment variable is assumed to contain a value which is substituted at run time.

The ACL enforcer in the client ensures that the permissions and resources specified in the ACL for the code are provided and no additional permissions/resources are allowed.

The ACL enforcement may be static or dynamic. In static enforcement, the enforcement can be done fully before the code is executed and no enforcement is required while the code is running. In dynamic enforcement, the enforcement must be performed while the code is being executed. If the CA itself verifies the ACL and guarantees that they will not be exceeded by the code, then the ACL enforcement function may not be required in the client system.

Figure 4 shows the data structures used by the ACL enforcer. The Runtime Physical Resources Table (RPRT) 300 contains a row for each resource. The resource name 300, resource attribute 310, maximum consumption rate 315 and maximum amount 320 are copies of the corresponding fields from the PRT 200. The actual consumption rate 325 and the actual use 330 fields are used to keep track of the actual consumption rate and the actual consumption, respectively, of the code during run time. The Runtime Logical Resource Table (RLRT) 350 contains a row for each required logical resource. The RLRT is a copy of the LRT, with an additional flag that indicates for each valid combination of parameters, whether the combination is allowed by the ACL manager or not. The Logical Resource Name 355 is a copy of the corresponding fields of the LRT 250 and the parameter list 360 points to a list of runtime parameter entries (RPE) 365. The parameter type 370, parameter value 375 values of the runtime parameter entries are copies of the corresponding fields in the LRT. The nextPE 380 points to the next runtime parameter entry 365 and the allow 385 field may be set to YES or NO indicating whether this runtime parameter entry 365 is allowed or not.The ACL enforcer also keeps track of the code start time 395.

Figure 5 shows how logical resource permissions are enforced by the ACL enforcer. This path is invoked whenever the code makes a call to an external function. In step 410, the ACL enforcer locates the number of parameters, their values, and the name of the function being invoked. The exact method for doing this is implementation dependent; for example, in Java, these are located on the operand stack. In step 415, the ACL enforcer locates the row for this function in the RLRT 350 and locates the first RPE 365 with allow set to YES. If the function name is not found, or there is no such RPE 365, the ACL enforcer proceeds to step 455 where it exits indicating that the call is not allowed. In step 420, the ACL enforcer locates the value of the first parameter and makes that the current parameter. In step 425, the ACL enforcer checks if the value of the current parameter is allowed by the RPE 365. If the value is allowed, the ACL enforcer checks in step 430 if there are more parameters. If there are, in step 435 the ACL enforcer sets the current parameter to the next parameter and returns to step 425. If no more parameters are found in step 430, the ACL enforcer proceeds to step 450 and returns with an indication that the call is allowed.

If the test for allowability fails in step 425, the ACL enforcer proceeds to step 445 where it checks the RLRT 350 to find another RPE 365 with allow 385 set to YES. If there is no such RPE, the ACL enforcer proceeds to step 455 and exits indicating that the call is not allowed. If there is such an RPE 365, the ACL enforcer proceeds to step 420.

Figure 6 shows how physical requirements are enforced by the ACL enforcer. The ACL enforcer is invoked in step 500 before allocating the resource. Two parameters, the amount of the resource requested (REQAMT) and the estimated time of consumption (COMPT) are provided. For disk I/O, the REQAMT is the amount of disk I/O and the COMPT is the estimated time for the I/O to complete. In step 505, the ACL enforcer locates the row in the RPRT 300 for this resource and checks whether the maximum amount 320 is specified and the actual use 330 plus REQAMT exceeds the maximum amount 320. If so, it returns a failure in step 527 indicating that the consumption is not allowed. If the maximum amount 320 is not exceeded, in step 510 the ACL enforcer computes the projected consumption rate of this resource as (Actual use 330 + REQAMT)/(current time - code start time 395 + COMPT) . In step 515, the ACL enforcer checks to see if the maximum consumption rate 315 is specified and if the projected consumption rate is greater than the maximum consumption rate 315. If the maximum consumption rate 315 is not specified or the projected consumption rate is not greater, the ACL enforcer returns with an indication that the consumption is allowed. If the projected consumption rate is greater, the ACL enforcer in step 530 computes the required delay for performing this operation as (actual use 300 + REQAMT)/maximum consumption rate 315 -(current time - code start time 395 -COMPT) and returns the required delay with an indication that the consumption has to be delayed for the computed delay.

In step 550, the ACL enforcer is invoked after the consumption of the resource with a parameter specifying the amount of resource consumed (CONSAMT). The ACL enforcer then updates the actual consumption rate 325 and the actual use 330.

It is also possible for different users to be allocated different resources and permissions in the recipient system for the same code. This can be done either at the time the code is installed, by the ACL manager by looking at the privileges given to the different users and combining that with the resources and permissions allowed to the code. In this

case the set of resources and permissions for each user would have to be stored separately. During code execution the resources and permissions would be enforced on a per-user basis. Alternatively the resources and permissions can be determined during execution of the code by the ACL enforcer by looking at the privileges given to the different users and combining that with the resources and permissions allowed to the code.

Figure 7 shows the pseudocode for the integrity verification, execution and enforcement initialization operations of the client system. It should be understood that the various tables, lists flags and other data structures described herein are instantiated in the memory of the client system (e.g. in volatile random access memory, disk or a combination of both). As previously discussed, the ACL enforcer and ACL manager are preferably embodied as program code which is linked or incorporated into the operating system of the client system. Figure 8 shows pseudocode for the ACL manager. Figure 9 shows pseudocode for the ACL enforcer.

Now that the invention has been described by way of the preferred embodiment, various modifications and improvements will occur to those of skill in the art. Thus, it should be understood that the preferred embodiment has been provided as an example and not as a limitation.

In summary, there is described a form of authentication is provided wherein a trusted third party signs a certificate to identify the author of a program and to secure its integrity. The program code is encapsulated or otherwise associated with the certificate and an access control list (ACL). The access control list describes the permissions and resources required by the code. An enforcement mechanism which allocates system permissions and resources in accordance with the ACL. In a preferred embodiment, a code production system communicates with a certification agency, which is a trusted third party. The certification agency issues a certificate for the code and a certificate for the access list of that code. Once the certificate is issued it is not possible for any party to modify the code or access list without invalidating the certificate. The code and its ACL, along with their certificates are stored on a server. A client downloading the code or access list can verify the integrity of the code/access list and the system can enforce the access list such that the permissions and resources are not exceeded.

Claims

1. A method for distributing program code comprising the steps of providing to a recipient system a trusted third party certification, the trusted third party certification including a computer readable description of resources and permissions required for verified non-harmful operation of the code.

2. The method of Claim 1 wherein the proving comprising the step of encapsulating the certification with the program code.

3. The method of Claim 1 or 2 comprising the further steps of reading the certification by the recipient system; and, allocating resources and permissions of the recipient system so as not to exceed the permissions specified in the certification.

4. The method of Claim 1, 2 or 3 comprising the further steps of denying or granting the code access and permissions to resources of the recipient system in further accordance with user selected options associated with the certification.

5. The method of any of Claims 1 to 4 wherein the computer readable description of resources and permissions is provided to the recipient system in an encrypted form.

6. The method of any of Claims 1 to 5 wherein the certification further includes encrypted verification data and wherein the recipient system decrypts the verification data to verify the integrity of the description of the resources and permissions.

7. The method of any of Claims 1 to 6 wherein the descriptions of resources required includes data describing both a quantity of each resource to be used by the code and a maximum rate of consumption of each resource by the code.

8. The method of any of Claims 1 to 7 wherein the descriptions of permissions required includes data describing specific facilities of the recipient system to be accessed by the code.

9. The method of any of Claims 1 to 8 wherein the third party certification includes a description of the functionality of the code as verified by the third party.

10. The method of any of Claims 1 to 9 wherein the program code is an applet downloaded from a server as a program

object.

11. The method of any of Claims 1 to 10 wherein different users are allocated different resources and permissions in the recipient system for the same code.

12. The method of Claim 11 wherein the set of resources and permissions allowed to different users is determined during installation of the code or during execution of the code.

13. A method for distributing program code comprising the steps of:

providing a recipient system with an encrypted trusted third party certification of the program code, the trusted third party certification being encapsulated with the program code and including a computer readable description of resources and permissions required for verified non-harmful operation of the code;

reading the certification by the recipient system;

determining the integrity of the certification by the recipient system; and

only after the integrity has been verified;

allocating resources and permissions of the recipient system in accordance with user selected options so as not to exceed the permissions specified in the certification; and

executing the program code in accordance with the allocating;

wherein the descriptions of resources required includes data describing both a quantity of each resource to be used by the code and a maximum rate of consumption of each resource by the code and wherein the descriptions of permissions required includes data describing specific facilities of the recipient system to be accessed by the code.

14. A computing system, comprising:

an importation device for importing programs and data into the computing system;

an operating system for controlling the operation of the computing system;

access logic for extracting from the data and associating with a given program a computer readable description of resources required for verified non-harmful operation of the code, the access logic further including integrity checking logic for generating verification data indicative of the integrity of the computer readable description; and

enforcement logic coupled to the operating system and responsive to the verification data tracking and allocating for each of a number of resources, consumption and consumption rate within the recipient system so as not to exceed the allocations specified in the description; and

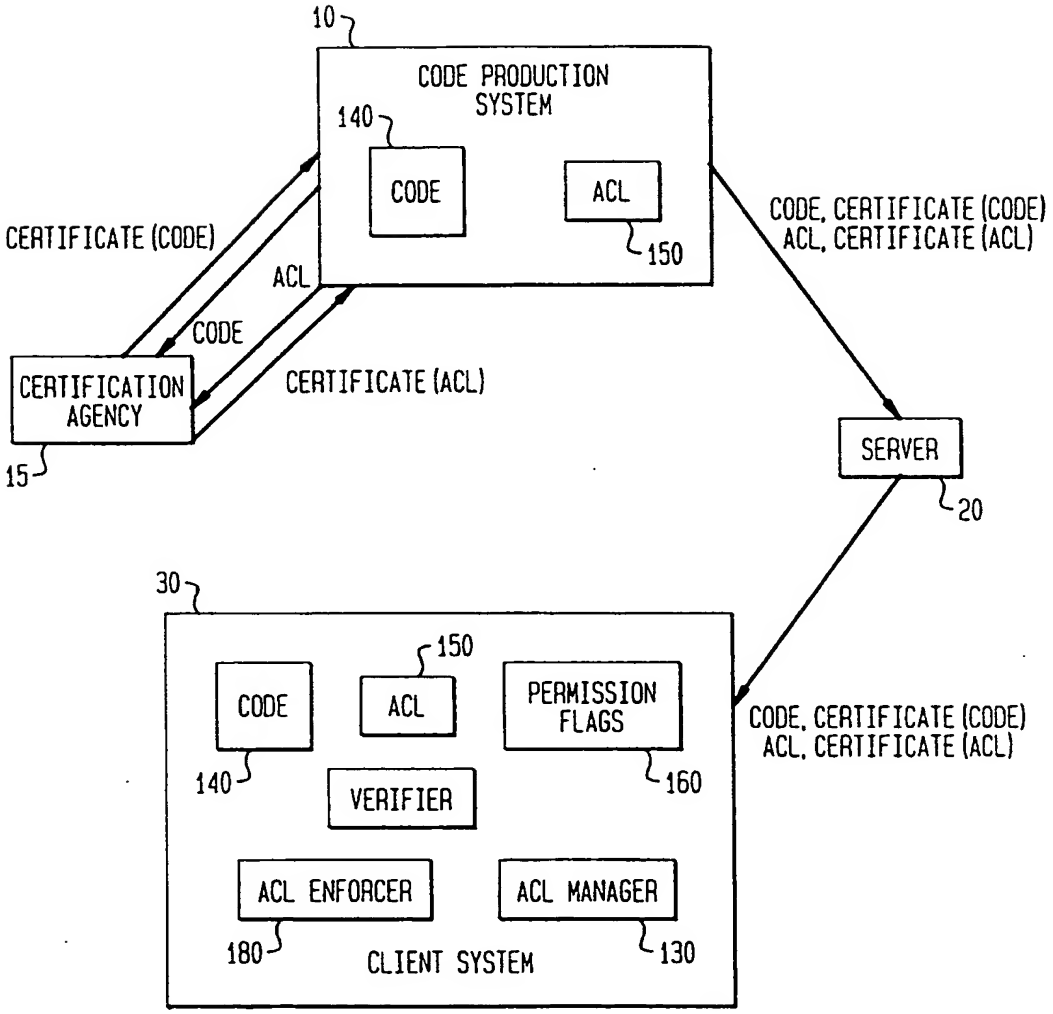
a processor for executing the program code in accordance with the allocating.

15. The system of Claim 14 further including a data structure stored in a random access memory and coupled to the enforcement logic, the data structure including at least a first field for tracking actual resources consumed and a second field for tracking rate of consumption of the resources, a third field for storing a limit on resource consumption derived from the description and a fourth field for storing limit on the rate of consumption of the resource derived from the description.

16. The system of Claim 15 wherein the access manager includes means for de-encapsulating the description from a package including the program code and means for decrypting the description.

17. The system of Claim 15 further wherein the enforcement logic includes means for denying or granting the code access and permissions to resources of the computing system in further accordance with user selected options associated with the description.

FIG. 1


10
CODE PRODUCTION SYSTEM
140
CODE
ACL
150
CERTIFICATE (CODE)
ACL
CODE
CERTIFICATE (ACL)
CERTIFICATION AGENCY
15
CODE, CERTIFICATE (CODE)
ACL, CERTIFICATE (ACL)
SERVER
20
30
150
CODE
ACL
PERMISSION FLAGS
140
160
VERIFIER
ACL ENFORCER
ACL MANAGER
180
130
CLIENT SYSTEM
CODE, CERTIFICATE (CODE)
ACL, CERTIFICATE (ACL)

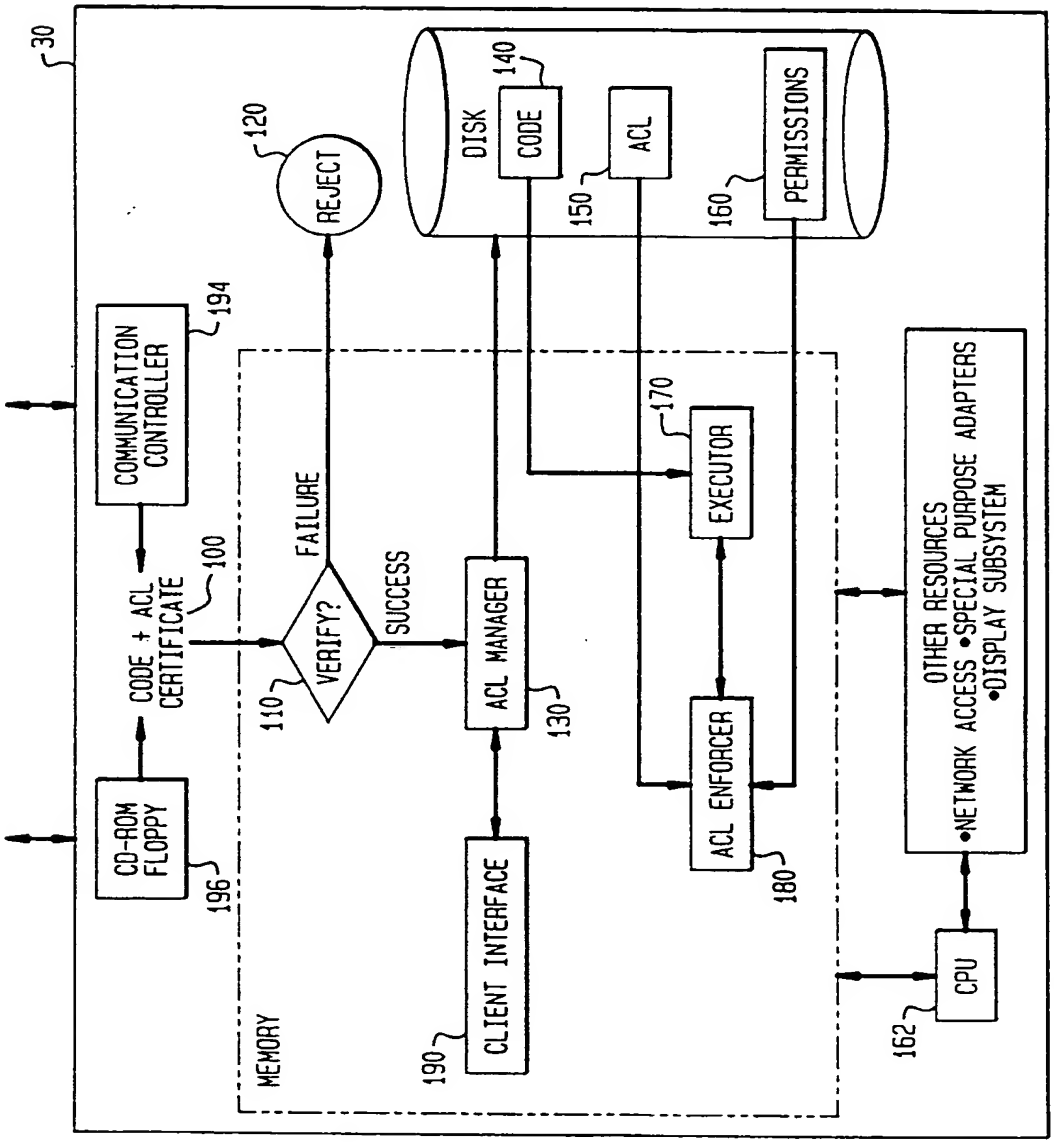

30
COMMUNICATION CONTROLLER
194
CODE + ACL CERTIFICATE
100
CD-ROM FLOPPY
196
MEMORY
VERIFY?
110
FAILURE
SUCCESS
120
REJECT
ACL MANAGER
130
CLIENT INTERFACE
190
DISK
CODE
140
ACL
150
PERMISSIONS
160
EXECUTOR
170
ACL ENFORCER
180
OTHER RESOURCES
•NETWORK ACCESS •SPECIAL PURPOSE ADAPTERS
•DISPLAY SUBSYSTEM
CPU
162

FIG. 2

FIG. 3

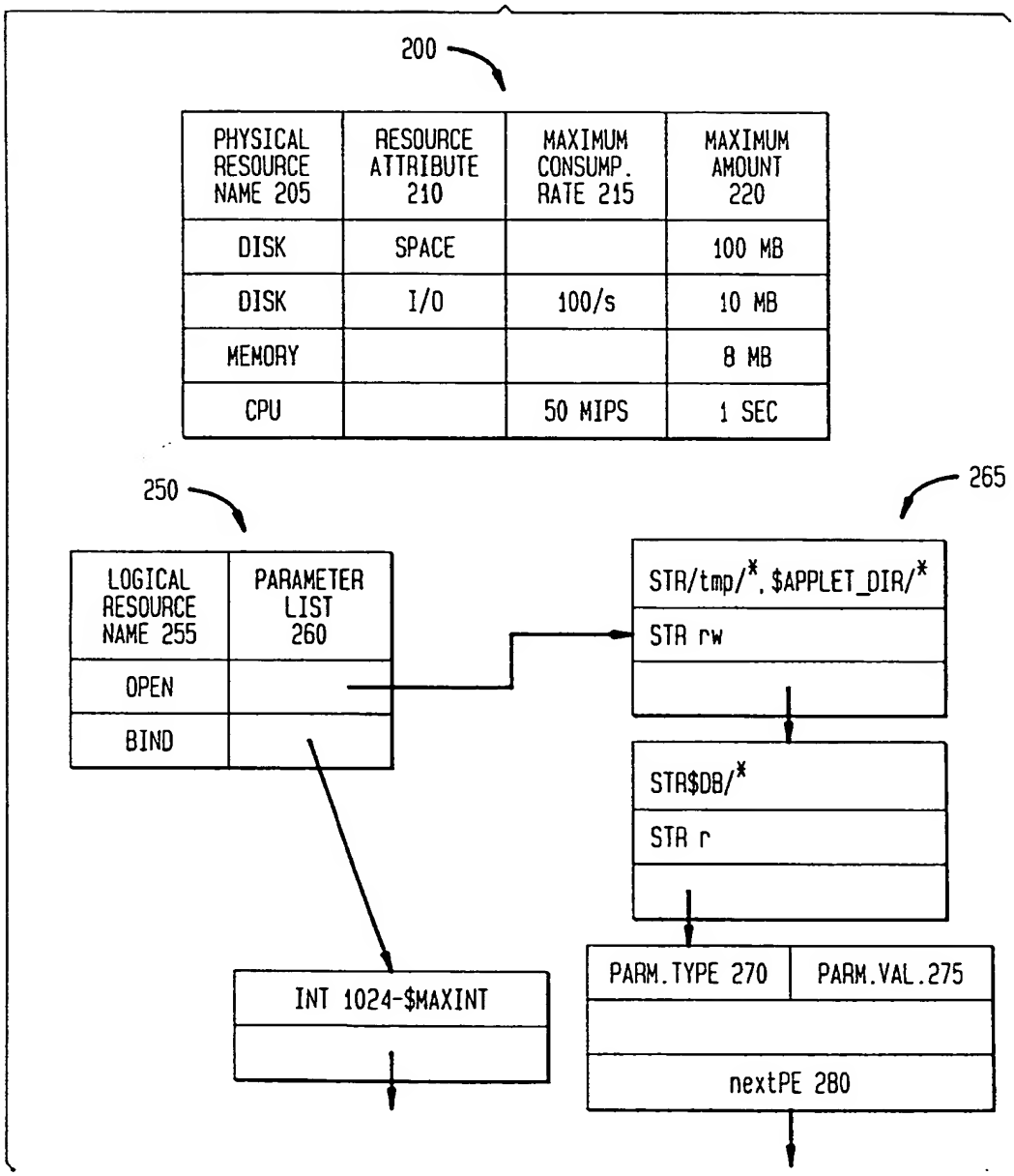

200
PHYSICAL RESOURCE NAME 205
RESOURCE ATTRIBUTE 210
MAXIMUM CONSUMP. RATE 215
MAXIMUM AMOUNT 220
DISK
SPACE
100 MB
DISK
I/O
100/s
10 MB
MEMORY
8 MB
CPU
50 MIPS
1 SEC
250
265
LOGICAL RESOURCE NAME 255
PARAMETER LIST 260
OPEN
BIND
STR/tmp/*.$APPLET_DIR/*
STR rw
STR$DB/*
STR r
INT 1024-$MAXINT
PARM.TYPE 270
PARM.VAL.275
nextPE 280

FIG. 4

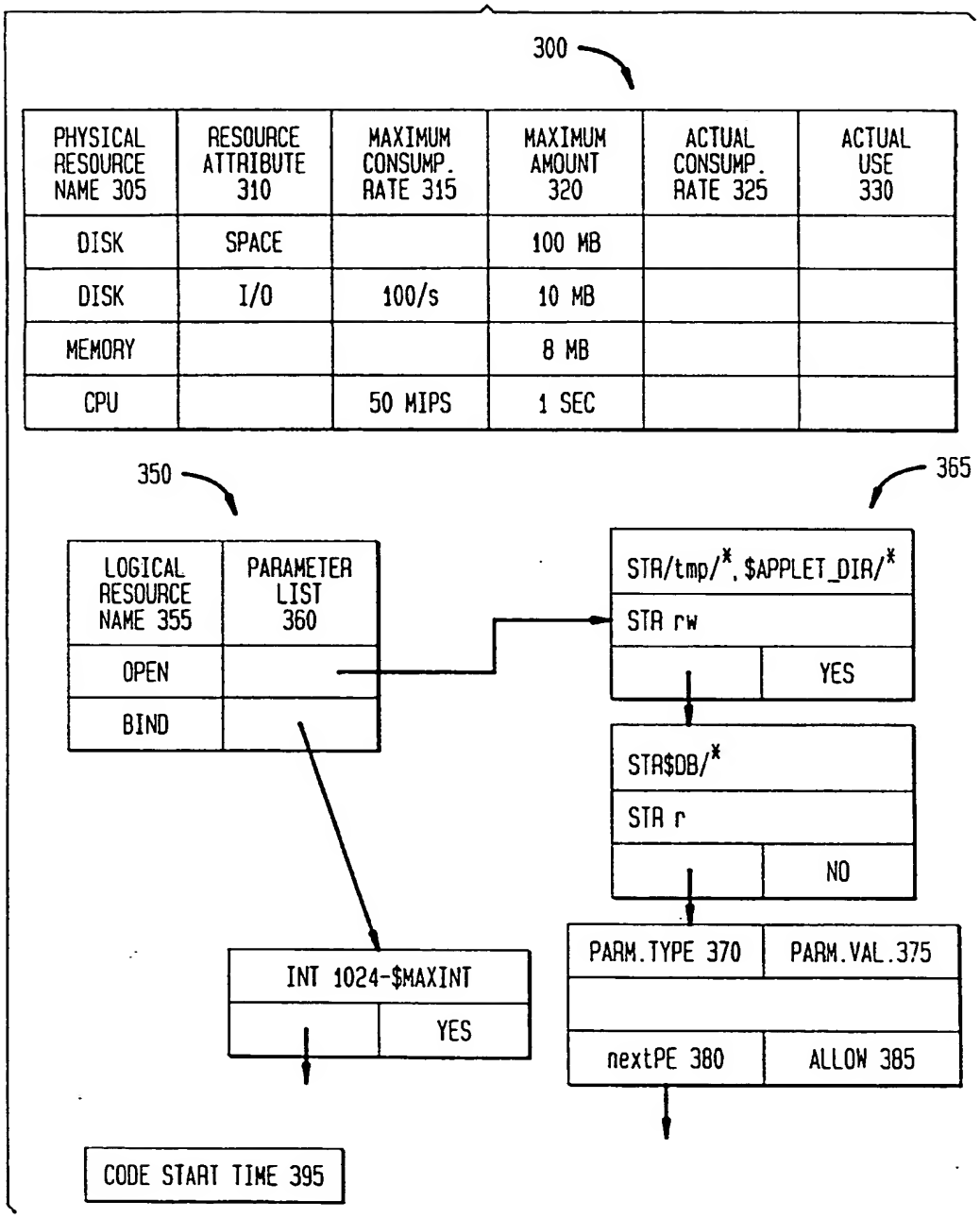

300
PHYSICAL RESOURCE NAME 305
RESOURCE ATTRIBUTE 310
MAXIMUM CONSUMP. RATE 315
MAXIMUM AMOUNT 320
ACTUAL CONSUMP. RATE 325
ACTUAL USE 330
DISK
SPACE
100 MB
DISK
I/O
100/s
10 MB
MEMORY
8 MB
CPU
50 MIPS
1 SEC
350
LOGICAL RESOURCE NAME 355
PARAMETER LIST 360
OPEN
BIND
365
STR/tmp/*, $APPLET_DIR/*
STR rw
YES
STR$DB/*
STR r
NO
INT 1024-$MAXINT
YES
PARM.TYPE 370
PARM.VAL.375
nextPE 380
ALLOW 385
CODE START TIME 395

FIG. 5

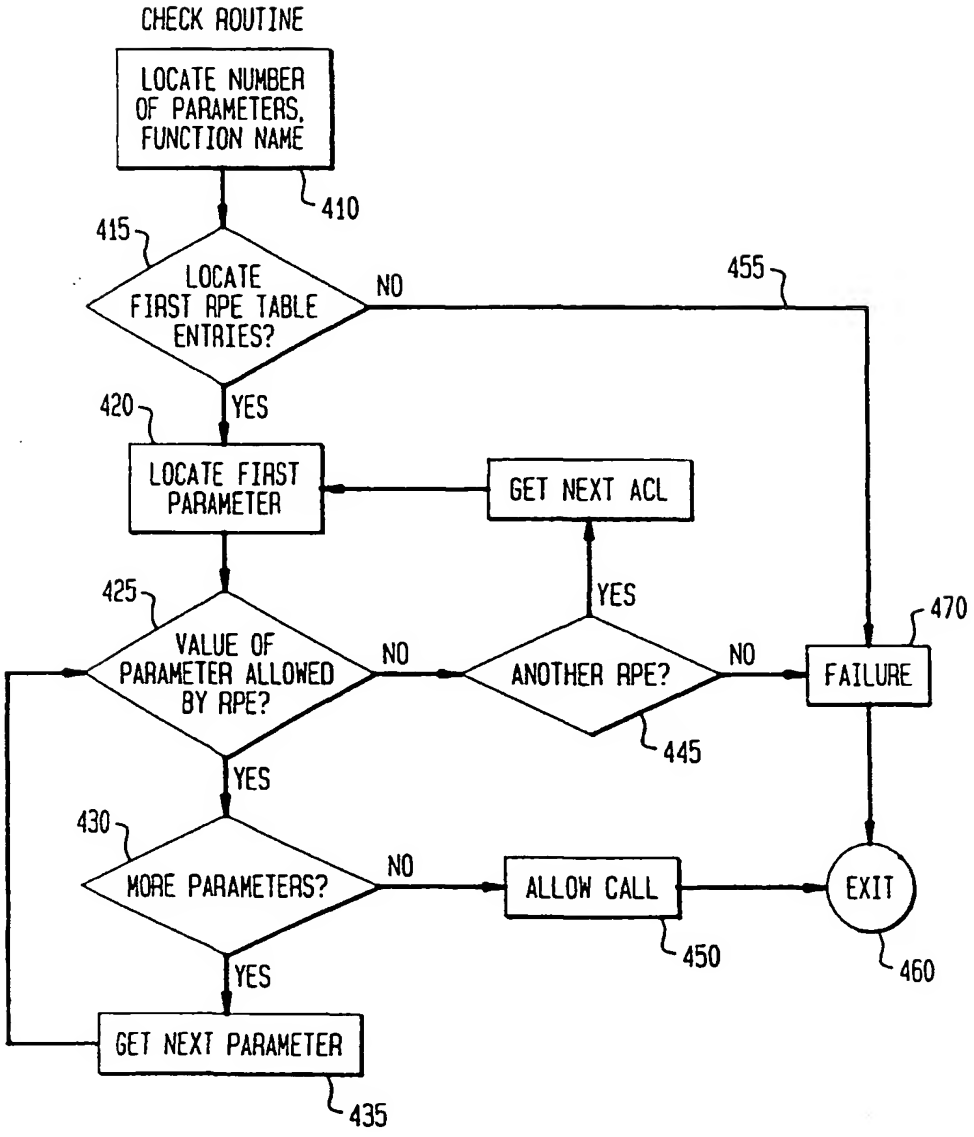

CHECK ROUTINE
LOCATE NUMBER OF PARAMETERS, FUNCTION NAME
410
415
LOCATE FIRST RPE TABLE ENTRIES?
NO
455
YES
420
LOCATE FIRST PARAMETER
GET NEXT ACL
425
VALUE OF PARAMETER ALLOWED BY RPE?
NO
YES
ANOTHER RPE?
NO
YES
445
470
FAILURE
430
MORE PARAMETERS?
NO
ALLOW CALL
EXIT
450
460
YES
GET NEXT PARAMETER
435

FIG. 6

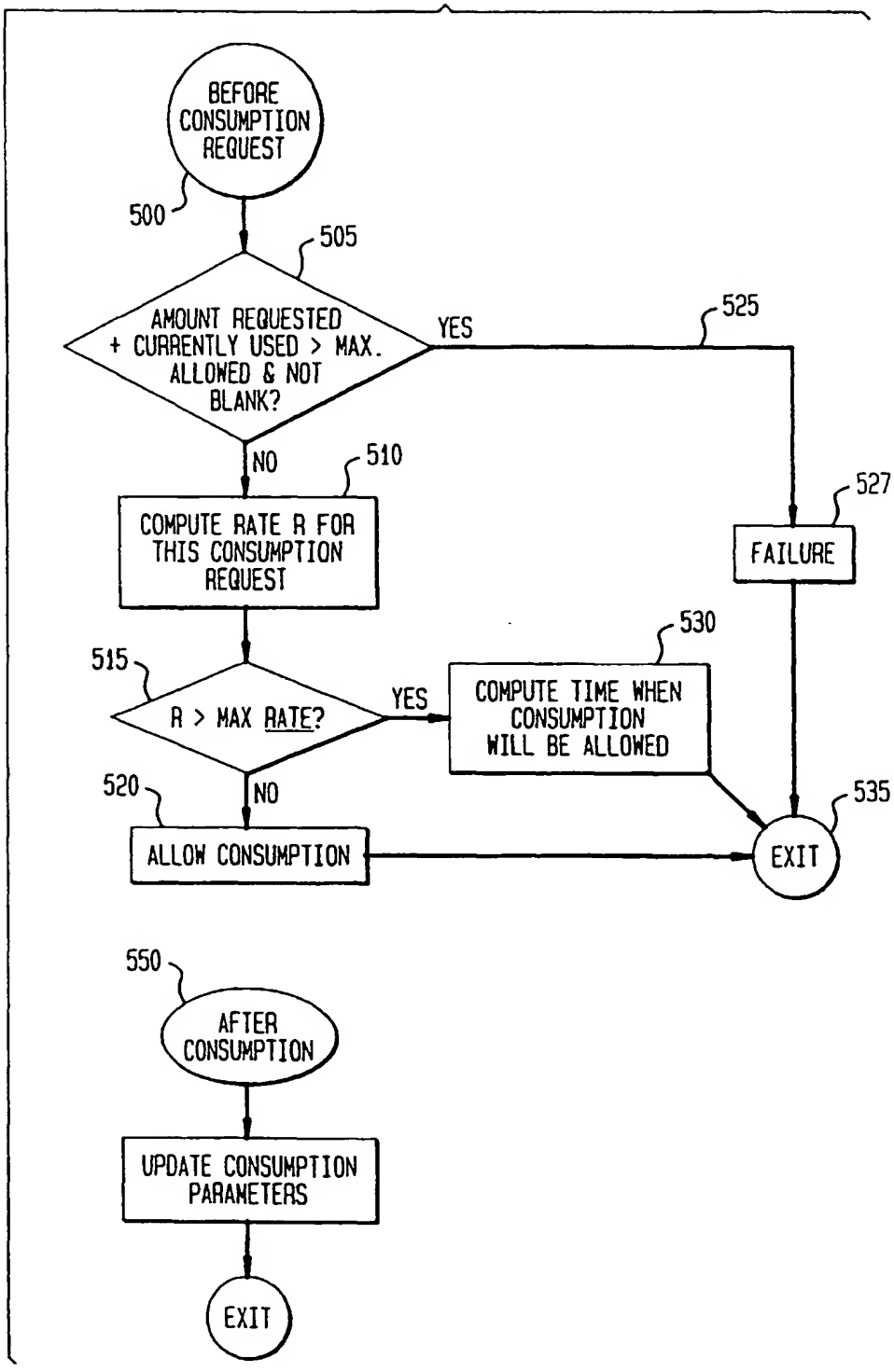

BEFORE CONSUMPTION REQUEST
500
505
AMOUNT REQUESTED + CURRENTLY USED > MAX. ALLOWED & NOT BLANK?
YES
525
NO
510
COMPUTE RATE R FOR THIS CONSUMPTION REQUEST
527
FAILURE
530
515
R > MAX RATE?
YES
COMPUTE TIME WHEN CONSUMPTION WILL BE ALLOWED
520
NO
535
ALLOW CONSUMPTION
EXIT
550
AFTER CONSUMPTION
UPDATE CONSUMPTION PARAMETERS
EXIT

FIG. 7

```
install_code:
------------

   Receive program code, ACL and encrypted hash;
   Decrypt using third party's public key.
   Compute hash from code and ACL;
   if computed_hash not equal decrypted_hash then
      Reject code;
      exit;
   invoke ACL manager;

execute_code:
------------

   Insert traps in code to verify_access whenever
resource is accessed;   Execute modified code;

verify_access:
------------

   Invoke ACL enforcer;
   If access permitted then
      continue;
   else if access delayed then
      delay program;
   else if access rejected then
      suspend program;
```

FIG. 8

```
If newly received code then
   Store code and ACL on disk;
Display ACL for program;
while more client input do
   read client input;
   Modify appropriate permission flags;
   Store modified permission flags on disk;
end
```

FIG. 9

```
check_logical_res_access:
-------------------------

   Locate RPE entries for this logical resource;
   while more RPE entries do
     while more parameters do
        if access to parameter not allowed then
           goto getNextRPE;
     end
     return access allowed;

   getNextRPE: get next RPE;
   end
   return access rejected;

check_physical_res_access:
--------------------------

   Locate RPRT entry for this physical resource;
   if Max_allowed non-blank and amt.requested + used >
max allowed then       return access rejected;
   Computed projected consumption rate;
   if projected consumption rate > max. consumption
rate then       Compute required delay;
     return access delayed and required delay;
  else
     return access allowed;
```

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 033 652 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
06.09.2000 Bulletin 2000/36

(51) Int Cl.7: **G06F 9/455**, G06F 1/00, G06F 9/46

(21) Application number: **00660043.1**

(22) Date of filing: **03.03.2000**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **03.03.1999 FI 990461**

(71) Applicant: **NOKIA MOBILE PHONES LTD.**
02150 Espoo (FI)

(72) Inventor: **Parkkinen, Jukka**
90580 Oulu (FI)

(74) Representative: **Brockman, Pertti Erik et al**
Kolster Oy Ab,
P.O. Box 148,
Iso Roobertinkatu 23
00121 Helsinki (FI)

(54) Method for downloading software from server to terminal

(57) The invention relates to a telephone system and a method for downloading software from a server (128) to a terminal (100, 102), the method comprising the steps of attaching to the software a certificate confirming the authenticity of the software and the loader; downloading the software from a source computer (134) to the server (128); downloading the software from the server (128) to the terminal (100, 102). In the method of the invention a first electronic signature confirming the authenticity of the software is attached to the software at the server (128). After the software is downloaded, a second electronic signature is generated at the terminal from the loaded software and the authenticity of the software is checked by comparing the first electronic signature with the second.

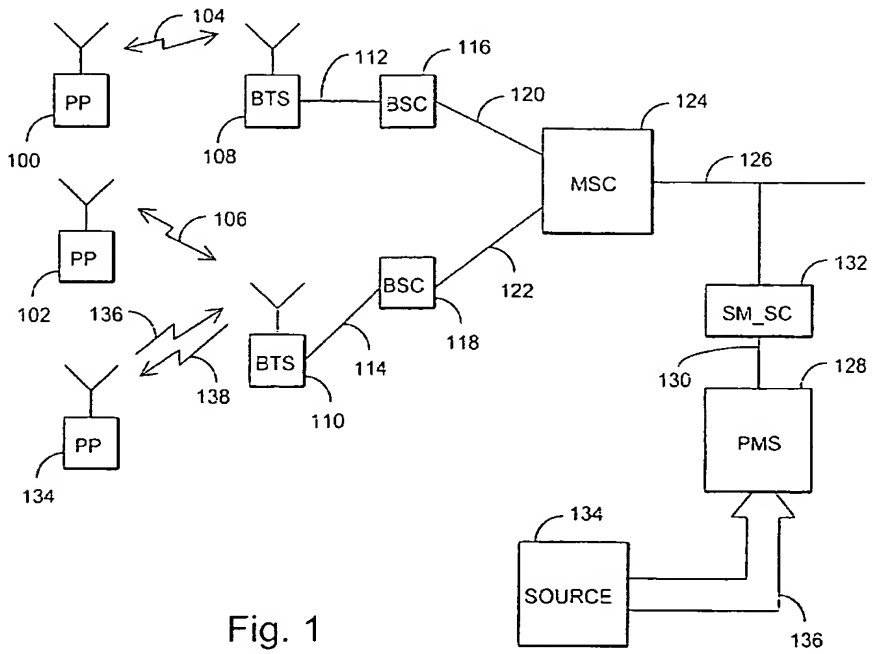


Fig. 1

EP 1 033 652 A2

Description

FIELD OF THE INVENTION

[0001] The invention relates to a method for downloading software from a server to a terminal in a telephone system comprising a plural number of terminals and a management system server that monitors and controls the operation of the terminals, a terminal of the system comprising means for storing one or more software.

BACKGROUND OF THE INVENTION

[0002] As radio telephone systems become increasingly common and their coverage areas grow - the systems often replacing those implemented by fixed line telephone connections - it has become necessary to develop telephone networks supporting radio telephone systems such as cellular radio systems. Such telephones are needed, for example, in areas where fixed line telephone connections do not exist, or in applications in which the terminal is in a place, for example in a moving vehicle, where connection to a fixed network is not easily available. The present invention can be applied particularly to systems implemented by means of cellular radio systems.

[0003] The systems and terminals involved include pay phones, so-called WLL (wireless local loop) terminals, payment terminals at points of sale and smart card terminals supporting transfer of money between a card and a bank.

[0004] The functions in current terminals are to a large extent implemented by means of various types of software. The terminal comprises a processor and memory into which the necessary software is stored. When the user selects a function, the software is read from the memory and carried out. In the designing of terminals, a compromise between the number of functions and the available memory capacity has been necessary. Due to reasons of cost, the size of the memory in the terminals cannot be infinitely increased, therefore the memory limits the number of the functions.

[0005] Let us study, by way of example, a pay phone system implemented by means of a radio system. The system comprises a plural number of pay phones, each communicating with base stations over a radio path. For the radio path and the base station, the terminals functioning as pay phones do not deviate in any way from conventional subscriber terminals. For collection of payments, the pay phones comprise a collection device that can typically be a payment card reading device. Numerous different payment cards are available, such as different types of credit cards, reloadable payment cards, bank cards, etc. In addition, the card types vary according to the card manufacturer and the company offering the card, and different facilities can be selected for one and the same card. Each card type requires the terminal to be provided with software supporting the card, i.e. a card application. The card application comprises the routines required for the terminal's user interface, for controlling the card and for performing a transaction, such as a payment.

[0006] To have card applications supporting all card types stored into the memory of a terminal reading a card would require such a large memory that the terminal would be expensive. Furthermore, the adding of new card applications to the terminal would require the software of the entire equipment to be changed at hardware maintenance.

[0007] Problems similar to those relating to pay phones also affect other wireless devices in which payment cards are read, such as reloading devices allowing electronic money to be loaded from a bank account to a payment card.

[0008] To solve the above problem, it is advantageous if software can be downloaded through the network when necessary, thereby allowing the terminal's memory to be optimally utilized. When a card is inserted into a terminal which does not have software corresponding to the card, the terminal can download the needed software to its memory through the network from a predetermined server.

[0009] This method has, however, its shortcomings. The use of software downloaded from a network involves risks that must be taken into account. It is important that the software to be downloaded is flawless and does not contain software viruses, for example, or other harmful elements. It is also important to be able to verify that the software is downloaded from the correct server and that it is manufactured by the correct software manufacturer. A defective software can cause malfunction in the terminal, such as unintended calls and transactions to wrong addresses.

BRIEF DESCRIPTION OF THE INVENTION

[0010] An object of the invention is therefore to provide a method and an apparatus implementing the method so as to allow the above problems to be solved. This is achieved with a method for downloading software from a server to a terminal, the method comprising the steps of attaching to the software a certificate confirming the authenticity of the software manufacturer and the loader; downloading the software from a source computer to the server; calculating a check sum for the software and the certificate; and downloading the software from the server to the terminal. The method of the invention further comprises the steps of adding the check sum confirming the authenticity of the software to the software at the server before the software is downloaded to terminals; generating a second check sum at the terminal from the downloaded software, after the software has been downloaded; and checking the authenticity of the software at the terminal by comparing the first check sum with the second.

[0011] The invention further relates to a telephone system comprising a plural number of terminals and a server monitoring and controlling the operation of the terminals, the server being arranged to calculate a check sum for the software and the certificate attached to the software; a terminal of the telephone system comprising means for storing one or more software, and the system comprising one or more source computers arranged to upload software to the server, the terminals being arranged to download the software from the server. In the telephone system of the invention the server is arranged to attach to the software a first check sum confirming the authenticity of the software before the software is downloaded to the terminals, and a terminal is arranged to generate a second check sum from the downloaded software, after the software has been loaded, and that the terminal is arranged to check the authenticity of the software by comparing the first check sum with the second.

[0012] The dependent claims relate to preferred embodiments of the invention.

[0013] The method and system of the invention provide several advantages. With the solution of the invention it is easy to ensure that the software is safe and that it is uploaded to the server from a safe source computer. The invention employs digital signature to ensure the authenticity of the software. Corresponding methods have earlier been applied only in connection with electronic mail transmissions.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] In the following the invention will be described in greater detail in connection with preferred embodiments and with reference to the accompanying drawings, in which

Figure 1 is a diagram illustrating a structure of a telephone system of the invention;
Figure 2 is a block diagram illustrating a structure of a terminal of a system according to the invention;
Figure 3 is a flow chart illustrating a method of the invention; and
Figure 4 is a flow chart illustrating the downloading of software.

DETAILED DESCRIPTION OF THE INVENTION

[0015] In the following the invention will be described, by way of example, with reference to a pay phone system implemented by applying a digital GSM mobile phone system, the invention not being, however, limited to the example. It is apparent that the solution of the invention can be modified to apply to telephone systems implemented by means of any other technology and comprising terminals which include functions operated by means of software applications.

[0016] Figure 1 illustrates a structure of a pay phone system implemented in a cellular radio network. The system comprises a plural number of pay phones 100-102, each connected via a radio path 104-106 to base stations 108-110. For the radio path or the base station, terminals operating as pay phones do not differ in any way from conventional subscriber terminals. The base stations 108-110 are typically connected to base station controllers 116-118, each controller controlling a plural number of base stations, via transmission lines 112-114 which can be implemented by means of optical cables, copper cables or link connections. The base station controllers 116-118, in turn, are connected via transmission lines 120-122 to a mobile services switching centre 124 which controls the operation of the base station controllers and transmits calls from the terminals to a fixed network or to other parts of the cellular radio system via transmission lines 126.

[0017] The pay phone system further comprises a management system server 128 which controls and monitors the operation of the pay phones 100-102. In the GSM system used as an example, a control equipment server 128 of the pay phone system is connected via an X.25 interface 130, for example, to a short message centre 132 which is, in turn, connected to GSM cellular networks and their mobile switching centres. The above description of the cellular radio system thus relates to the GSM system, but it is obvious that although the details of other systems vary from the above description, there are no essential structural differences. It should be noted that also in the GSM system the pay phone system can be implemented without the short message centre, by connecting the control equipment server 128 of the pay phone system to the cellular radio system by employing other prior art methods, such as a modem.

[0018] The system of the invention further comprises a source computer 134, such as a computer of the manufacturer of the software used in the terminals. The source computer 134 is connected to the server 128 via a telecommunications network 136, such as the Internet or a private network. Both the server and the source computer can be implemented as computer hardware having the required telecommunications characteristics and the appropriate software.

[0019] Figure 2 illustrates an example of a preferred embodiment of a pay phone according to the system of the invention. The pay phone of the invention comprises a cellular radio transceiver 200 and a control unit 204 which has a direct connection 202 to the transceiver 200 without a two-wire connection. The terminal of the invention further comprises a collection means 206 connected to the control unit 204. Depending on the application, the collection means can accept phone cards, credit cards or smart cards as means of payment. The terminal typically also comprises a dialling means 210 for dialling the desired telephone number, display unit 208 and an earpiece 212. The terminal can also comprise means 214 allowing a hands free facility, the

means comprising a speaker 216 and a microphone 218, and the necessary amplifiers. If desired, some or all of the above components can be directly integrated into the transceiver 200, or they can be implemented as separate means, although structurally possibly within the same casing.

[0020] The function of the transceiver unit 200 is to provide, when necessary, a radio connection to a base station to allow a call to be transmitted. The unit 200 also takes care of all operations (usually carried out by a mobile phone) concerning the maintenance of the radio path and the call.

[0021] The function of the control unit 204 is to control the pay phone. The control unit typically comprises a micro processor, fixed and reprogrammable memory circuits, multiplexing means and switches. The control unit controls the operations of other units included in the equipment, registers placed calls and takes care of debiting. The operational parameters of the pay phone are usually stored in the control unit's memory. Such telephone-specific parameters include telephone number, tariff data relating to the calls to be placed, language options on the telephone's display and volume of voice. Except for the inventive features described in the present application, the operation of the control unit does not basically differ from the operation of the control units of prior art pay phones.

[0022] The details of the terminal structure can also vary from the above description depending on the purpose of use of the terminal. For example, if the terminal is a payment terminal used at a point of sale, the device does not necessarily include audio parts such as a microphone or speaker. At its simplest, the terminal comprises a cellular radio transceiver, a control unit and collection means which can be structurally integrated with each other or, alternatively, they may be components detachable from one another and temporarily connected together for the duration of a call payment or a purchase transaction, for example.

[0023] The software needed by the terminal are stored into the memory of a control unit 204. The software concerned include software, or card applications, needed by various payment card alternatives. A card application comprises routines needed for the terminal's user interface, for controlling a card and for carrying out a card transaction, such as a payment.

[0024] Let us then study the method of the invention with reference to a flow diagram shown in Figure 3. As stated above, the system of the invention allows software to be downloaded to terminals, when necessary, from the system server. To ensure the authenticity of the software it is important that software can only be uploaded to the server from a source the authenticity of which has been confirmed. In the solution of the invention, each software supplier is therefore provided with a specific digital certificate that allows the software supplier, or the supplier's computer (hereinafter referred to as the source computer) from which the software is uploaded to the server, to be identified. The certificate is granted by a third party, such as the terminal manufacturer.

[0025] In step 300 of Figure 3, the software producer attaches a digital certificate confirming the authenticity of the software to the software to be transferred to a server. In step 302 the software is uploaded from the software producer's source computer via, for example, the Internet or another link to the network server which in this example is the server of the pay phone system. In a preferred embodiment of the invention, the server checks the source computer's certificate in connection with the downloading.

[0026] When software is downloaded to terminals, it is also essential that the software is downloaded from an official server agreed on in advance and not from a disturber that has connected to the network. It is therefore necessary that the origin of the software can be verified from the software. For this purpose the software is provided with an electronic signature at the server, the signature being attached to the software in step 306. In the preferred embodiment of the invention, the electronic signature is generated by calculating a check sum in step 304 for the software and the certificate and by attaching the check sum to the software in step 306, preferably by using encryption, thereby preventing any external party from corrupting the sum. The check sum itself can be calculated by applying methods known to those skilled in the art. One way of implementing the encryption is to use a public key and secret key encryption method. The electronic signature is attached to the software at the server in step 306 by using the server's secret key which outsiders do not know. The encrypted information can then be decrypted by using a public key at the terminal. In the solution of the invention, encryption methods known to those skilled in the art can be used.

[0027] In step 308 the terminal downloads the software needed from the server. After the terminal has downloaded the software, it checks the authenticity of the software in step 310 by calculating, similarly as at the server, the check sum of the downloaded software and the certificate attached to the software. The terminal then decrypts the encrypted electronic signature attached to the software at the server in step 312 by using the server's public key. As a result of the decryption, the check sum calculated at the server is obtained. The terminal compares the check sum it has calculated with that calculated at the server in step 314, the result of the comparison allowing the terminal to decide the authenticity. If the check sums match, the software is authentic (step 316), but if the check sums do not match, the source of the software is not authentic (step 318) and the software cannot be taken into use.

[0028] Let us then study an example of a situation where the above described downloading of the software cannot be carried out; this is illustrated in a flow diagram of Figure 4. In step 400 the user has inserted a card into a card reader 206 of a terminal. In step 402 the terminal

checks the different functions of the card, for example, any credit card alternatives included. If several options are available, the user gets to select the function to be used. The routine then proceeds to step 406 to check whether an application required by the selected function is included in the terminal's memory. The application keeps record of the applications available in its memory at a particular moment. If the application is in the memory, it can be started in step 408.

[0029] If the application is not in the terminal's memory, the routine proceeds to step 410 to check whether the application is in the management system's server. Information about the applications that can be downloaded from the server can be stored either in the terminal, or the terminal can request the information from the server. If the application cannot be found from the management system, the function is rejected in step 412 and the user is asked to give a new one, provided that the card contains several functions.

[0030] If the application is on the management system's server, the terminal asks in step 414 the amount of memory required by the application. The terminal then checks in step 416 whether the amount of memory required by the application is available. If there is not enough memory available, an application to be removed from the memory is selected and removed in step 418 so as to release memory for the new application. The terminal can let the user select the application to be removed or, alternatively, the terminal can make the decision on the basis of a predetermined criterion. One criterion is to keep recently used applications and to remove an application that has been unused for the longest.

[0031] The terminal then informs in step 420 the server of a free memory area where the application should be placed. For example, the terminal can inform a memory area 312, shown in Figure 3, to be available for the application. The management system's server downloads in step 422 the application to the memory area informed by the terminal. The application is then ready to be taken into use in step 424.

[0032] In another alternative embodiment the management system's server does not control the placing of the application into the terminal's memory, but only transmits the application to the terminal which then places the application into its memory.

[0033] In addition to payment card applications, a downloadable software can comprise facilities transferred in an electronic form, such as timetable information or tickets.

[0034] Method steps associated with the terminal of the invention can be advantageously implemented by software at the terminal's control unit 204. The connection to the management system's server required by the method can be advantageously provided by means of a data call connection. A data call is a call type that is available in digital radio networks; it corresponds to a modem connection in analog systems.

[0035] At the management system's server and in the software manufacturer's source computer the functions of the invention can be advantageously implemented by means of software.

[0036] Although the invention is described above with reference to an example shown in the attached drawings, it is apparent that the invention is not restricted to it, but can vary in many ways within the inventive idea disclosed in the attached claims.

Claims

1. A method for downloading software from a server (128) to a terminal (100, 102), the method comprising the steps of

   attaching to the software a certificate confirming the authenticity of the software manufacturer and the loader;
   uploading the software from a source computer (134) to the server (128);
   calculating a check sum for the software and the certificate; and
   downloading the software from the server (128) to the terminal (100, 102),

   characterized in that the method further comprises the steps of

   attaching the check sum confirming the authenticity of the software to the software at the server (128) before the software is downloaded to terminals;
   generating a second check sum at the terminal from the downloaded software, after the software has been downloaded; and
   checking the authenticity of the software at the terminal by comparing the first check sum with the second.

2. A method according to claim 1, **characterized** in that the authenticity of the software is always checked at the terminal (100, 102) when the software is carried out.

3. A method according to claim 1, **characterized** in that the method comprises the generating of an electronic signature at the server (128) by calculating for the software and the certificate a common check sum which is encrypted by means of a secret key of the server.

4. A method according to claim 3, **characterized** in that the encryption of the secret key is decrypted at the terminal (100, 102) by means of a public key of the server (128).

5. A method according to claim 1, **characterized** in that the terminal (100, 102) detects that a payment card is inserted into the terminal's card reader (206) and the user has selected an application, and that the terminal

checks whether the software needed for implementing the application can be found in the terminal's memory, and
sends the server (128) a loading request comprising information about the software needed, and that the server
sends the terminal the software needed, and that the terminal
stores the software into its memory.

6. A telephone system comprising

a plural number of terminals (100, 102) and
a server (128) monitoring and controlling the operation of the terminals, the server (128) being arranged to calculate a check sum for software and for a certificate attached to the software;
a terminal of the telephone system comprising means (204) for storing one or more software, and the system comprising
one or more source computers (134) arranged to upload software to the server, the terminals (100, 102) being arranged to download software from the server,

characterized in that

the server is arranged to attach to the software a first check sum confirming the authenticity of the software before the software is downloaded to the terminals, and that
a terminal is arranged to generate a second check sum from the downloaded software, after the software has been loaded, and that the terminal is arranged to check the authenticity of the software by comparing the first check sum with the second.

7. A system according to claim 6, **characterized** in that the terminal is arranged to always check the authenticity of the software when the software is carried out.

8. A system according to claim 6, **characterized** in that the server is arranged to generate an electronic signature by calculating for the software and the certificate a common check sum and to encrypt the calculated check sum by means of a secret key of the server.

9. A system according to claim 6, **characterized** in that the terminal is arranged to decrypt the encryption of the electronic signature by means of a public key of the server.

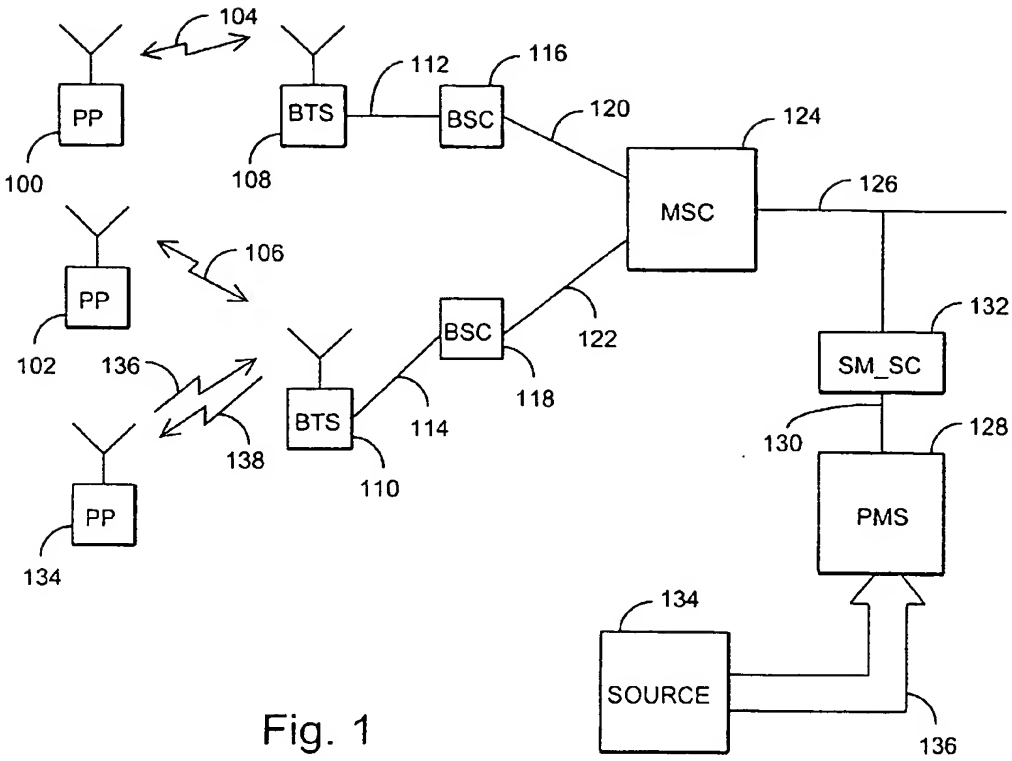

104
PP
100
BTS
108
112
BSC
116
120
MSC
124
126
PP
102
106
136
138
BTS
110
114
BSC
118
122
132
SM_SC
130
128
PMS
PP
134
134
SOURCE
136

Fig. 1

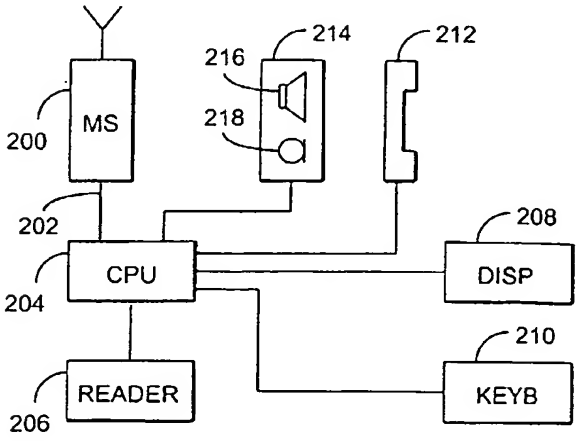

MS
200
216
214
212
218
202
208
CPU
204
DISP
210
READER
206
KEYB

Fig. 2

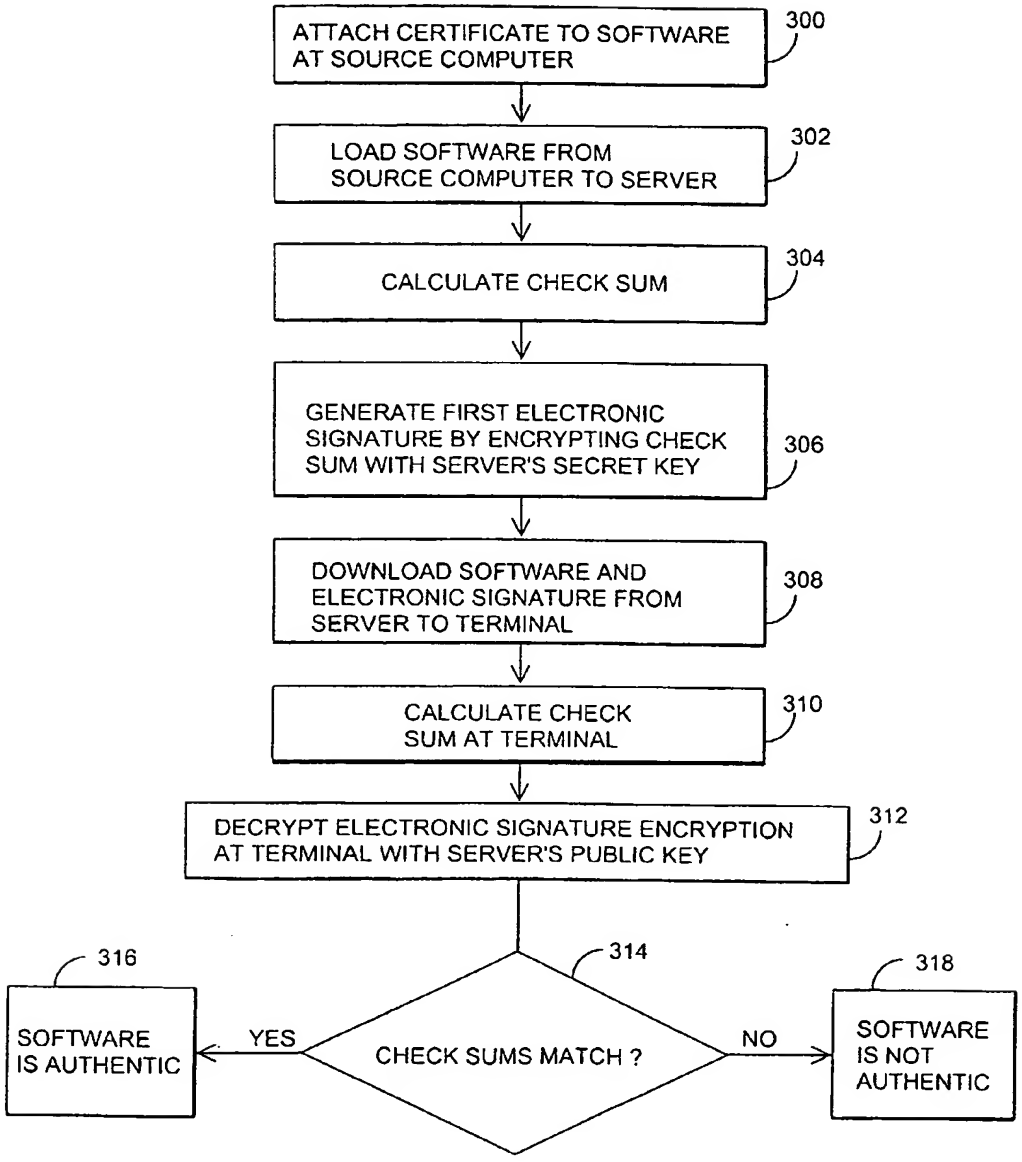

ATTACH CERTIFICATE TO SOFTWARE AT SOURCE COMPUTER
300
LOAD SOFTWARE FROM SOURCE COMPUTER TO SERVER
302
CALCULATE CHECK SUM
304
GENERATE FIRST ELECTRONIC SIGNATURE BY ENCRYPTING CHECK SUM WITH SERVER'S SECRET KEY
306
DOWNLOAD SOFTWARE AND ELECTRONIC SIGNATURE FROM SERVER TO TERMINAL
308
CALCULATE CHECK SUM AT TERMINAL
310
DECRYPT ELECTRONIC SIGNATURE ENCRYPTION AT TERMINAL WITH SERVER'S PUBLIC KEY
312
CHECK SUMS MATCH ?
314
YES
NO
SOFTWARE IS AUTHENTIC
316
SOFTWARE IS NOT AUTHENTIC
318

Fig. 3

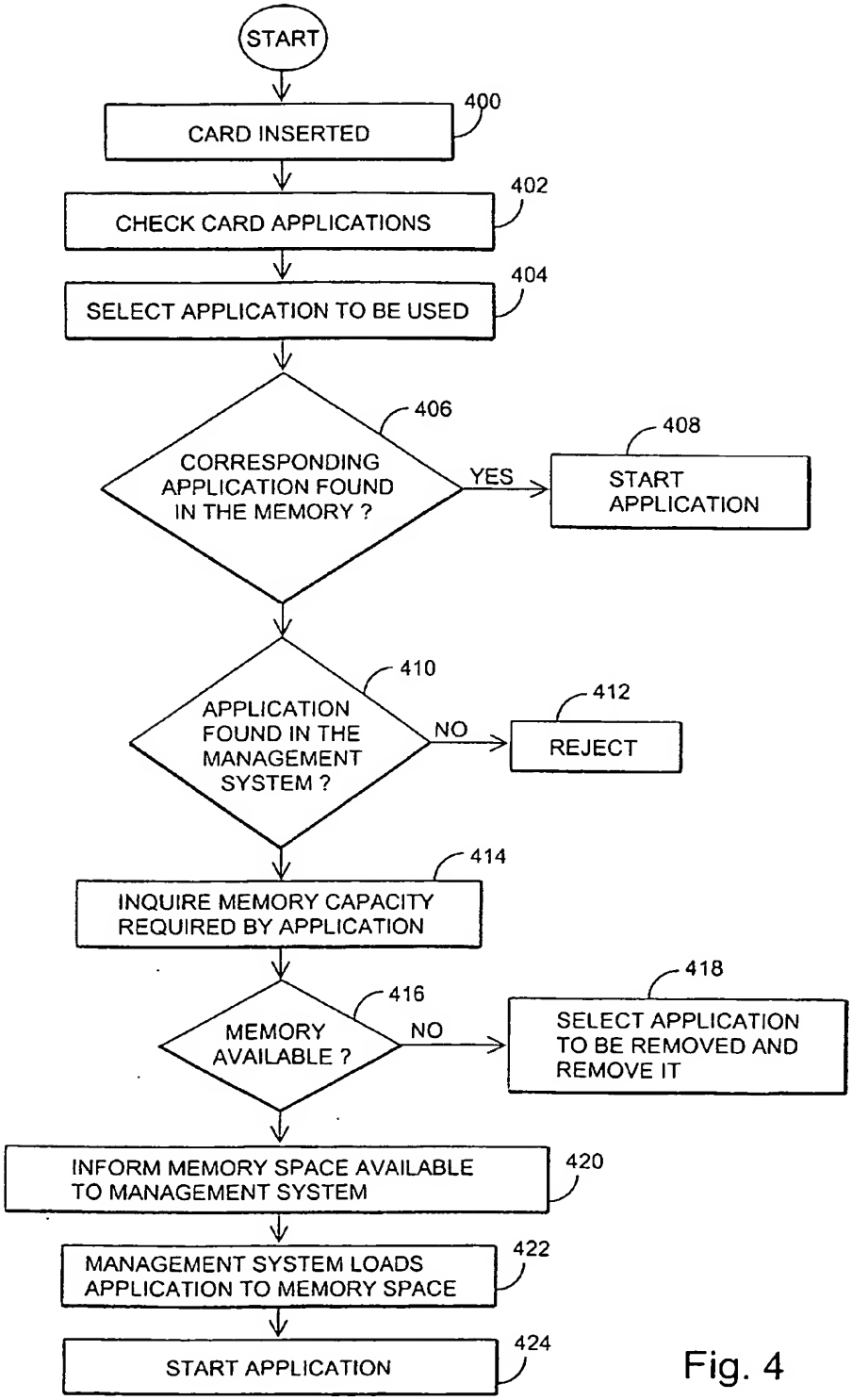

START
CARD INSERTED
400
CHECK CARD APPLICATIONS
402
SELECT APPLICATION TO BE USED
404
CORRESPONDING APPLICATION FOUND IN THE MEMORY ?
406
YES
START APPLICATION
408
APPLICATION FOUND IN THE MANAGEMENT SYSTEM ?
410
NO
REJECT
412
INQUIRE MEMORY CAPACITY REQUIRED BY APPLICATION
414
MEMORY AVAILABLE ?
416
NO
SELECT APPLICATION TO BE REMOVED AND REMOVE IT
418
INFORM MEMORY SPACE AVAILABLE TO MANAGEMENT SYSTEM
420
MANAGEMENT SYSTEM LOADS APPLICATION TO MEMORY SPACE
422
START APPLICATION
424

Fig. 4